



Centro Universitario de Ciencias Exactas e Ingenierías.

Ciencias de la computación

Seminario de Uso, Adaptación y Explotación de Sistemas Operativos, Becerra Velázquez Violeta del Rocío

Becerra Velázquez Violeta del Rocío

Hernandez Lomeli Diego Armando

219750396

INNI- Ingeniería en informática

D02

Actividad de aprendizaje 11

2.4 Productor consumidor

2023/10/24

Tabla de contenido

| | |
|--|--------------------------------------|
| Introducción: | 3 |
| Objetivo: | 3 |
| Instrucciones. | 3 |
| a) Investigue el problema del productor-consumidor con sus diferentes escenarios..... | 3 |
| b) Una vez verificado en que consiste debe programarlo utilizando semáforos. | 3 |
| c) El lenguaje de programación a utilizar es a su elección, por lo tanto, puede ser en modo grafico o consola..... | 4 |
| d) Para su implementación debe tener en consideración los siguientes puntos: | 4 |
| e) Genere un reporte donde incluya la explicación del problema, la solución que propone y la explicación de su código (incluya imágenes)..... | 6 |
| f) Realice un video con la demostración del programa funcionando, súbalo en algún sitio en la nube y no olvide proporcionar permisos públicos de visualización (si no se puede ver el video solo tendrá la mitad de los puntos de esta actividad)..... | 23 |
| g) Incluya el enlace al video en su reporte. | 23 |
| h) Genere una conclusión sobre lo que ha aprendido en esta actividad. | 24 |
| i) Entregue su tarea al buzón de tareas antes de la fecha de término. | ¡Error! Marcador no definido. |
| Ilustración 1 constructor del Productor consumidor | 6 |
| Ilustración 2 espacio en vista del productor y consumidor..... | 6 |
| Ilustración 3 espacio representativo del buffer gráficamente..... | 7 |
| Ilustración 4 indicación de estados de cada actor..... | 8 |

Productor consumidor

Introducción:

Investigar el problema del productor-consumidor con sus diferentes escenarios, ver la importancia de la exclusión mutua en el problema de la concurrencia, evitando así interbloqueo e inanición, una vez estudiado el problema programar la solución utilizando semáforos.

Objetivo:

El estudiante aplicará una solución por medio de semáforos al problema del productor-consumidor implementando algoritmos de programación con lenguajes de alto nivel.

Instrucciones.

a) Investigue el problema del productor-consumidor con sus diferentes escenarios.

El problema del productor y el consumidor demuestra el uso de la exclusión mutua y el interbloqueo, este mismo involucra a mínimo 1 productor y 1 consumidor, algunas implementaciones consideran n productores y n consumidores, n productores y 1 consumidor, o viceversa, independientemente de la dirección de la implementación, siempre se necesitará de 1 buffer que sea el contenedor de los datos por proteger, las reglas de acceso a la escritura son bastante simples, pues se requiere que solo 1 (productor o consumidor) acceda al buffer a la vez, para ello se puede hacer uso de diferentes técnicas para exclusión mutua, y para que el productor pueda generar recursos debe existir espacio vacío en el buffer (no lleno) y para que el consumidor pueda tomar los recursos, el buffer no debe estar vacío (no vacío).

Algunas implementaciones de este problema se componen de 2 hilos de ejecución diferentes, donde 1 de ellos se encarga de la producción de nuevos elementos y el otro se encarga de consumir los elementos del productor. Durante la ejecución se necesita un mismo origen/colección de datos para operar, pero accediendo a la misma de forma alternada comunicando los hilos involucrados para indicar cuando es posible realizar una acción y bajo condiciones específicas cada uno, esto pretende desacoplar al productor del consumidor.

En esta ocasión no usaremos semáforos (planteamiento indicado en clase) y haremos que tanto productor y consumidor accedan 1 a la vez de forma alternada con 50% de probabilidad de ser elegido.

b) Una vez verificado en que consiste debe programarlo utilizando semáforos.

c) El lenguaje de programación a utilizar es a su elección, por lo tanto, puede ser en modo grafico o consola.

| Productor consumidor | | | |
|-------------------------|---------------------|---|---|
| Lenguajes y tecnologías | Entradas necesarias | Salidas esperadas | Descripción de funcionalidad |
| Javascript HTML | Ninguna | Visuales: - Lista (representación del buffer): en ella se mostrará el contenido más reciente del buffer (cada actualización a los datos alterará la vista). - Buffer sobreescritura en el índice actual del productor o consumidor con su carácter representativo (☹ o 🍔) | <p>Crear una instancia de clase “ProductorConsumidor”, cada instancia genera un buffer de 25 espacios rellenos con el carácter “☹” indicándolo como espacio libre, a su vez, se generará un nuevo elemento a la lista (en vista) por cada nueva entrada en el buffer.</p> <p>El “ProductorConsumidor” contiene 1 productor y 1 consumidor que alternaran turnos de manera aleatoria (con limite a 5 turnos consecutivos) para sobrescribir su carácter representativo en el buffer, si sus reglas lo permiten.</p> <p>Cuando el productor o consumidor cambian de estado se verá reflejado en sus espacio de vista.</p> |

d) Para su implementación debe tener en consideración los siguientes puntos:

1. Existen un solo un productor y un solo consumidor.
2. Se cuenta con un “contenedor” con capacidad para 25 elementos, en el cual el productor colocará y el consumidor retirará elementos.
3. El “contenedor”, lógicamente es un buffer circular y acotado, es decir al llegar a la última casilla (25) comenzará nuevamente en la casilla 1 (los espacios deben estar perfectamente señalados).
4. El producto puede ser: números, caracteres especiales, letras, hamburguesas, galletas, etc.
5. Solo puede ingresar uno a la vez al contenedor.
6. Para que el Productor pueda entrar, debe haber espacio en el contenedor y no estar el Consumidor dentro.
7. Para que el Consumidor pueda entrar, debe existir producto y no estar el productor dentro.
8. En la pantalla debe aparecer:
 - i. La información del productor, es decir, mostrar si está dormido, trabajando, cuando intente ingresar al contenedor, etc.

- ii. La información del consumidor, dormido, trabajando, cuando intente ingresar, etc.
 - iii. Mensajes que indiquen en todo momento, quien está trabajando, o quien intenta trabajar, o si está dormido.
9. Deben manejarse tiempos aleatorios para dormir al productor y al consumidor.
 10. Al “despertar” intentará producir y/o consumir respectivamente, verificando que pueda hacerlo según sus condiciones.
 11. Al entrar al buffer podrán producir y/o consumir de forma finita (de 1 a 5 elementos).
 12. El productor colocará elementos en orden, comenzando con la primera casilla, y continuando desde la última posición donde se quedó.
 13. El consumidor quitará elementos en orden, comenzando también por la primera casilla y continuando en la última donde quedo.
 14. El programa terminara al presionar la tecla escape.

e) Genere un reporte donde incluya la explicación del problema, la solución que propone y la explicación de su código (incluya imágenes).

1. Existen un solo un productor y un solo consumidor.

```
class ProductorConsumidor {
  constructor() {
    this.pivot = 50;
    this.bufferSeccion = document.getElementById("buffer");
    this.buffer = [];
    for (let index = 0; index < 25; index++) {
      this.buffer.push("🍷");
      let node = document.createElement("li");
      node.id = `spaceBuffer_${index}`;
      node.classList.add('card');
      node.classList.add('spaceBuffer');
      node.innerText = `🍷`;
      //Generación de espacio en vista
      this.bufferSeccion.appendChild(node);
    }
    this.productor = new Productor(this.buffer);
    this.consumidor = new Consumidor(this.buffer);
  }
}
```

El productor consumidor es quien contiene a cada uno de los actores definidos y al buffer al que se accederá. al final de la *ilustración 1* se muestra su construcción, al final se puede ver cómo están definidos tanto consumidor como productor.

Ilustración 1 constructor del Productor consumidor

El productor y el consumidor poseen un espacio reservado en la interfaz gráficas para indicar su estado actual y los accesos con modificaciones al buffer consecutivos

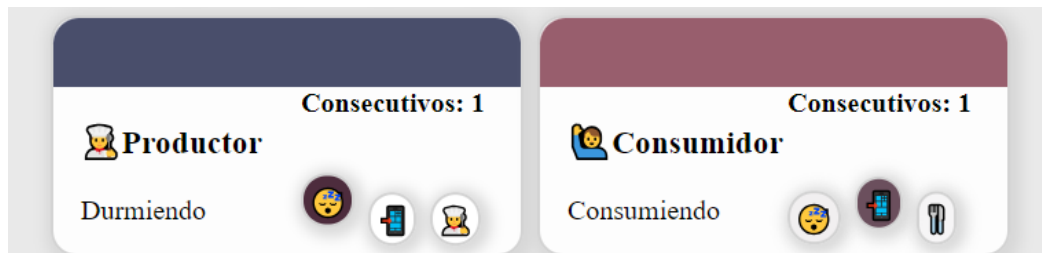


Ilustración 2 espacio en vista del productor y consumidor

2. Se cuenta con un “contenedor” con capacidad para 25 elementos, en el cual el productor colocará y el consumidor retirará elementos.

El contenedor es originalmente un Array de 25 elementos generado en cada instancia de “ProductorConsumidor” (Revisar elemento de la lista anterior)

Visualmente el buffer es una lista con disposición de elementos de tabla, cada elemento representa un espacio dentro del buffer que se alternará con los caracteres (🍷 o 🍷) según el actor que haya hecho la sobrescritura siendo que “🍷” es vacío y “🍷” es producido.

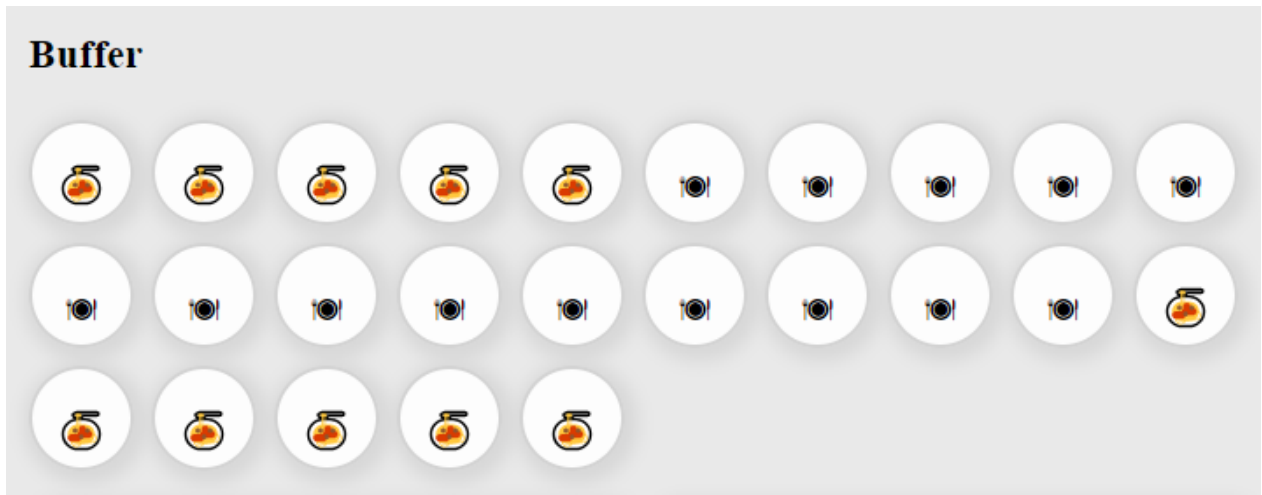






Ilustración 3 espacio representativo del buffer gráficamente

3. El “contenedor”, lógicamente es un buffer circular y acotado, es decir al llegar a la última casilla (25) comenzará nuevamente en la casilla 1 (los espacios deben estar perfectamente señalados).

Cada actor realiza su recorrido en el buffer según su propio índice de continuidad, si este índice supera al tamaño del buffer se reiniciará a la primera posición para evitar el desbordamiento.


4. El producto puede ser: números, caracteres especiales, letras, hamburguesas, galletas, etc.

Los elementos que pueden existir en el buffer son  o  donde “” es para el espacio consumido y “” es para el espacio con un objeto producido

5. Solo puede ingresar uno a la vez al contenedor.

El acceso al buffer es alternado entre los 2 actores con un 50% de probabilidad de entrar siempre y cuando no se superen los 5 elementos creados o producidos de manera consecutiva por cada actor en caso de tener 5 elementos producidos o consumidos consecutivamente, se pasará inmediatamente el turno al actor que haya estado dormido durante las interacciones con el buffer anteriores.

6. Para que el Productor pueda entrar, debe haber espacio en el contenedor y no estar el Consumidor dentro.

El Productor sabe que puede entrar al buffer si se encuentra un elemento “” dentro del buffer, si no se encuentra, se considera lleno y no puede producir (No aumentará el contador de elementos producidos consecutivamente en este caso).

En caso de no poder producir, se mantendrá el estado del actor como “durmiendo”.

7. Para que el Consumidor pueda entrar, debe existir producto y no estar el productor dentro.

El Consumidor sabe que puede consumir si existe por lo menos 1 elemento “🍌” dentro del buffer, si no se encuentra, se considera vacío y no puede consumir (No aumentará el contador de elementos consumidos consecutivamente en este caso).

En caso de no poder consumir, se mantendrá el estado del actor como “durmiendo”.

8. En la pantalla debe aparecer:

- I. La información del productor, es decir, mostrar si está dormido, trabajando, cuando intente ingresar al contenedor, etc.
- II. La información del consumidor, dormido, trabajando, cuando intente ingresar, etc.
- III. Mensajes que indiquen en todo momento, quien está trabajando, o quien intenta trabajar, o si está dormido.

Los estados de cada actor se ven directamente reflejados en cada espacio correspondiente, siendo que cuenta con 3 estados generales que saltan cuando su acción respectiva es utilizada y regresan a su lugar cuando se abandona ese estado.

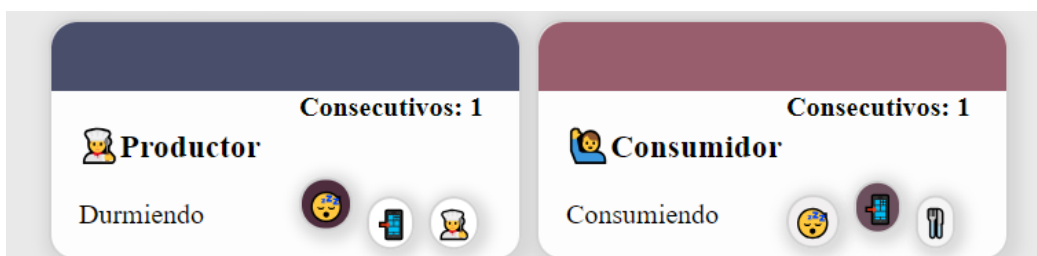


Ilustración 4 indicación de estados de cada actor

9. Deben manejarse tiempos aleatorios para dormir al productor y al consumidor.

Al tener alternancia aleatoria el tiempo en que cada actor duerme es impredecible, en ocasiones se generarán los 5 turnos consecutivos para cada actor y en otras se puede alternar entre 1 elemento consumido y 1 elemento producido entre los actores.

10. Al “despertar” intentará producir y/o consumir respectivamente, verificando que pueda hacerlo según sus condiciones.

Cada que un actor deja su estado “dormido” e intenta producir o consumir se actualizará gráficamente su sección correspondiente y acudiendo solo si es capaz y tratando de sobrescribir según las condiciones impuestas en el punto 6 y 7 de esta lista.

11. Al entrar al buffer podrán producir y/o consumir de forma finita (de 1 a 5 elementos).

Los elementos máximos que cada actor puede modificar consecutivamente son máximo 5, si se intenta modificar un sexto elemento, se pasará el turno inmediatamente al actor contrario.

12. El productor colocará elementos en orden, comenzando con la primera casilla, y continuando desde la última posición donde se quedó.

El productor cuenta con 1 índice interno para su propia continuidad en el buffer, este es independiente al consumidor y se reinicia en 0 cuando su valor supera el tamaño del buffer.

13. El consumidor quitará elementos en orden, comenzando también por la primera casilla y continuando en la última donde quedo.

El consumidor cuenta con 1 índice interno para su propia continuidad en el buffer, este es independiente al productor y se reinicia en 0 cuando su valor supera el tamaño del buffer.

14. El programa terminara al presionar la tecla escape.

Se puede detener la ejecución del algoritmo presionando la tecla “esc” pero se puede reanudar presionando el btn “reiniciar” de la interfaz gráfica.

Planteamiento del algoritmo

Para resolver este problema se generan 3 clases:

- 1 para productor
- 1 para consumidor
- 1 como máster de productor consumidor (nombrada “ProductorConsumidor”)

Las clases para productor y consumidor son prácticamente iguales pero conteniendo diferentes caracteres para indicar su acción dentro del buffer y una condición diferente para sobrescribir el buffer (explicada en la lista de especificaciones).

Cada uno de estos actores escribirá dentro de su propio espacio en interfaz gráfica su estado actual y sobrescrituras consecutivas, también será capaz de modificar el buffer directamente en sus datos e interfaz gráfica (cada vez que modifique).

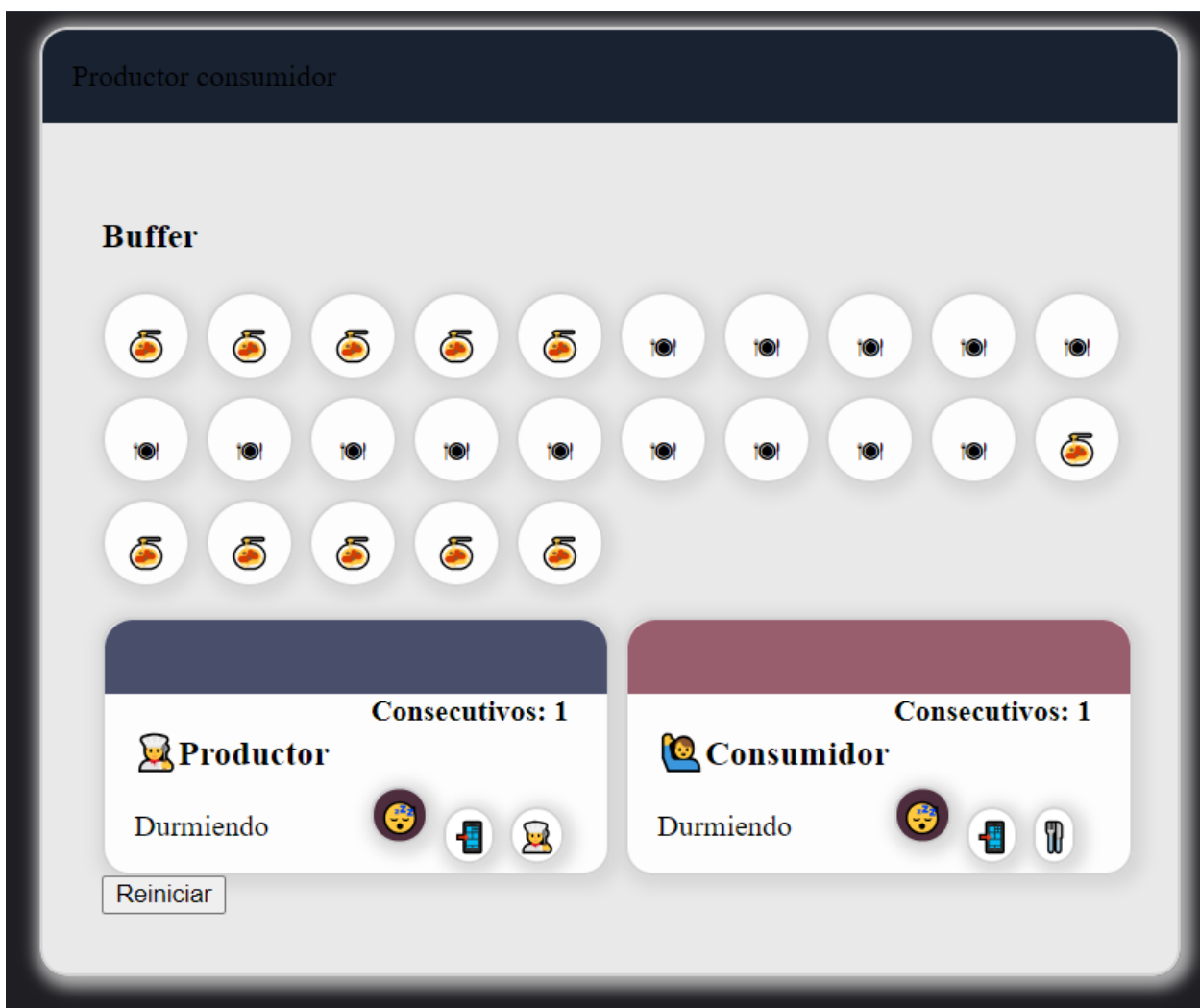
La clase master “ProductorConsumidor” se encargará de preparar los elementos para las clases anteriores y la interfaz visual (solo el espacio de los buffers) al ser construida una nueva instancia, también es contenedor principal del buffer. Además es quien decide que actor debe tomar lugar en la iteración actual, es decir, cada vez que se intenté acceder al buffer, la instancia de “ProductorConsumidor” se encargará de decidir quien es quien accederá al buffer (siguiendo las especificaciones de la lista de especificaciones).

Aspectos generales

La rutina se ejecuta gracias al método “window.requestAnimationFrame” que recibe una función como si se tratará de recursividad, esta misma función nombrada “initRuntime” asíncrona, con ello permite colocar espera de resolución a otras funciones, esto permite que existan pausas implícitas/arbitrarias en acciones en el código utilizadas para mostrar los estados al usuario.

La segunda parte de la implementación es sobre maquetación html, inicialmente esta parte no es funcional, solo gráfica y prepara los espacios necesarios para que el código en “js” pueda generar los elementos necesarios e interactuar con los que no necesitan una inicialización fuera de la maquetación base.

El resultado final de la implementación es el siguiente:



El código de implementación se muestra a continuación

Interfaz gráfica.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Productor consumidor</title>
    <script src="ProductorConsumidor.js"></script>
    <style>
      .grid {
        display: grid;

        grid-gap: 0.5em;
      }
      .twocolumns {
        grid-template-columns: repeat(2, minmax(50%, 1fr));
      }
      .tencolumns {
        grid-template-columns: repeat(10, 3em);
        grid-template-rows: repeat(3, 3em);
      }

      .modern-list {
        list-style: none;
        padding-left: 0px;
      }
      .buffer {
      }
      .spaceBuffer {
        padding: 1em;
        display: flex;
        align-content: center;
        justify-content: center;
        border-radius: 50% !important;
      }
      .card {
        background-color: rgba(255, 255, 255, 0.9);
        border-color: #fff;
        border: 2px solid lightgray;
        box-shadow: 2px 2px 8px 4px #d3d3d3d1;
        border-radius: 15px;
        overflow: hidden;
```

```

}
.header {
  background-color: #192231;
}
.tooltip .tooltiptext {
  visibility: hidden;
  width: 120px;
  background-color: black;
  color: #fff;
  text-align: center;
  border-radius: 6px;
  padding: 5px 0;

  /* Position the tooltip */
  position: absolute;
  z-index: 1;
}

.tooltip:hover .tooltiptext {
  visibility: visible;
}
main {
  width: 80%;
}
body {
  display: flex;
  justify-content: center;
  width: 100%;
  background-color: #1f1f23;
}
.p-2 {
  padding: 2em;
}
.p-1 {
  padding: 1em;
}
.ps-0 {
  padding-top: 0em;
}
.w-100 {
  width: 100%;
}
.w-50 {
  width: 50%;
}

```

```
.headerProductor {
  background-color: #494e6b;
  overflow: hidden;
  height: 2.5em;
  border-top-left-radius: 15px;
  border-top-right-radius: 15px;
}

.headerConsumidor {
  background-color: #985e6d;
  overflow: hidden;
  height: 2.5em;
  border-top-left-radius: 15px;
  border-top-right-radius: 15px;
}

.row {
  display: flex;
}

.elementoEstado {
  background-color: rgba(255, 255, 255, 0.9);
  border-color: #fff;
  border: 2px solid lightgray;
  box-shadow: 2px 2px 8px 4px #d3d3d3d1;
  border-radius: 15px;
  padding: 0.2em;
  transform: translateY(85%) !important;
  transition: 0.5s;
}

.seccionEstados {
  margin-bottom: 0px;
}

.elementoEstadoActivo {
  background-color: #4d2c3d;
  border-color: #fff;
  border: 2px solid lightgray;
  box-shadow: 2px 2px 8px 4px #d3d3d3d1;
  border-radius: 15px;
  padding: 0.2em;
  transform: translateY(50%) !important;
  transition: 0.5s;
}

.enfasis {
  margin-block-start: 1em;
  margin-block-end: 1em;
  margin-inline-start: 0px;
```

```

        margin-inline-end: 0px;
        font-weight: bold;
    }
</style>
</head>
<body>
    <main id="main" class="card">
        <div class="header p-1">Productor consumidor</div>
        <article class="p-2">
            <!-- Espacio dedicado para el buffer -->
            <h3>Buffer</h3>
            <ul id="buffer" class="grid tencolumns modern-list buffer"></ul>
            <div class="w-100 grid twocolumns">
                <div class="card">
                    <div class="headerProductor"></div>
                    <section class="p-1 ps-0" id="seccionProductor">
                        <div class="grid twocolumns">
                            <div>
                                <h3>&img alt="truck icon" data-bbox="338 418 358 433"/>Productor</h3>
                                <span id="estadoProductor"> Estado </span>
                            </div>
                            <div>
                                <span id="consecutivosProd" class="enfasis">
                                    Consecutivos
                                </span>
                                <ul class="grid modern-list row seccionEstados">
                                    <li id="prodEstSleep" class="tooltip elementoEstado">
                                        &img alt="sleeping person icon" data-bbox="338 580 358 595"/>
                                        <span class="tooltiptext">Durmiendo</span>
                                    </li>
                                    <li id="prodEstAccesing" class="tooltip elementoEstado">
                                        &img alt="person at computer icon" data-bbox="338 650 358 665"/>
                                        <span class="tooltiptext">Accediendo</span>
                                    </li>
                                    <li id="prodEstProd" class="tooltip elementoEstado">
                                        &img alt="factory icon" data-bbox="338 720 358 735"/>
                                        <span class="tooltiptext">Produciendo</span>
                                    </li>
                                </ul>
                            </div>
                        </div>
                    </section>
                </div>
            </div>
            <!-- Fin espacio productor -->
            <div class="card">

```

```
<div class="headerConsumidor"></div>
<section class="p-1 ps-0" id="seccionConsumidor">
  <div class="grid twocolumns">
    <div>
      <h3><img alt="User icon" data-bbox="338 165 358 185"/> Consumidor</h3>
      <span id="estadoConsumidor"> Estado </span>
    </div>
    <div>
      <span id="consecutivosCons" class="enfasis">
        Consecutivos
      </span>
      <ul class="grid modern-list row seccionEstados">
        <li id="consEstSleep" class="tooltip elementoEstado">
          <img alt="Sleeping person icon" data-bbox="338 325 358 345"/>
          <span class="tooltiptext">Durmiendo</span>
        </li>
        <li id="consEstAccesing" class="tooltip elementoEstado">
          <img alt="Accessing document icon" data-bbox="338 395 358 415"/>
          <span class="tooltiptext">Accediendo</span>
        </li>
        <li id="consEstCons" class="tooltip elementoEstado">
          <img alt="Fork icon" data-bbox="338 465 358 485"/>
          <span class="tooltiptext">Consumiendo</span>
        </li>
      </ul>
    </div>
  </div>
</section>
</div>
<!-- Fin espacio Consumidor -->
<div class="row">
  <button onclick="reInitRuntime()">Reiniciar</button>
</div>
</article>
</main>
</body>
</html>
```

Código de fondo.

```

class ProductorConsumidor {
  constructor() {
    this.pivot = 50;
    this.bufferSeccion = document.getElementById("buffer");
    this.buffer = [];
    for (let index = 0; index < 25; index++) {
      this.buffer.push("🍷");
      let node = document.createElement("li");
      node.id = `spaceBuffer_${index}`;
      node.classList.add(`card`);
      node.classList.add(`spaceBuffer`);
      node.innerText = `🍷`;
      //Generación de espacio en vista
      this.bufferSeccion.appendChild(node);
    }
    this.productor = new Productor(this.buffer);
    this.consumidor = new Consumidor(this.buffer);
  }

  /* Decidir si debe entrar el productor o el consumidor */
  decideAction() {
    this.productor.escribirConsecutivos();
    this.consumidor.escribirConsecutivos();
    let iniciarConsumidor = () => {
      this.productor.inRow = 0;
    };
    let iniciarProductor = () => {
      this.consumidor.inRow = 0;
    };
    if (this.productor.inRow > 4) {
      //Si supera las 5 seguidas devuelve al siguiente
      iniciarConsumidor();
      return this.consumidor;
    }
    if (this.consumidor.inRow > 4) {
      //Si supera las 5 seguidas devuelve al siguiente
      iniciarProductor();
      return this.productor;
    }
    //Validar primero la secuencia de seguidos de cada actor
    let rangeDecide = Math.random() * 100;
    //Dependiendo de la aleatoriedad decidimos cual devolver
    if (rangeDecide > this.pivot) {

```



```
    //Reescritura de seguidos
    iniciarConsumidor();
    return this.consumidor;
  }
  //Reescritura de seguidos
  iniciarProductor();
  return this.productor;
}
}

class Productor {
  constructor(buffer) {
    this.buffer = buffer;
    this.indexProd = 0; //Se reinicia al llegar a 25
    this.producto = "🍷";
    this._inRow = 0;

    this.fieldState = document.getElementById("estadoProductor");

    this.prodEstSleep = document.getElementById("prodEstSleep"); //Representación
    de estado actual
    this.prodEstAccesing = document.getElementById("prodEstAccesing");
    //Representación de estado actual
    this.prodEstProd = document.getElementById("prodEstProd"); //Representación
    de estado actual
    this.consecutivosProd = document.getElementById("consecutivosProd");
    //Representación de estado actual
  }

  async producir() {
    removeAddClass(this.prodEstSleep, "elementoEstado", "elementoEstadoActivo");
    await this.cambiarTextoEstado("Accediendo a buffer");
    removeAddClass(this.prodEstSleep, "elementoEstadoActivo", "elementoEstado");
    removeAddClass(
      this.prodEstAccesing,
      "elementoEstado",
      "elementoEstadoActivo"
    );
    if (!this.buffer.includes("🍷")) {
      //no se puede producir
      await this.cambiarTextoEstado("Durmiendo sin espacio");
      removeAddClass(
        this.prodEstAccesing,
        "elementoEstadoActivo",
        "elementoEstado"
      );
    }
  }
}
```

```

    );
    //Dormido
    removeAddClass(
        this.prodEstSleep,
        "elementoEstado",
        "elementoEstadoActivo"
    );
    return;
}
await this.cambiarTextoEstado("Produciendo");
removeAddClass(
    this.prodEstAccesing,
    "elementoEstadoActivo",
    "elementoEstado"
);
//Producido
this.buffer[this.indexProd] = this.producto;
//Escritura en vista
document.getElementById(`spaceBuffer_${this.indexProd}`).innerText =
    this.producto;

this.indexProd = this.indexProd >= 24 ? 0 : this.indexProd + 1;
removeAddClass(this.prodEstProd, "elementoEstado", "elementoEstadoActivo");
this.addConcurrence();

await this.cambiarTextoEstado("Durmiendo");
removeAddClass(this.prodEstProd, "elementoEstadoActivo", "elementoEstado");
//Dormido
removeAddClass(this.prodEstSleep, "elementoEstado", "elementoEstadoActivo");
//console.log(this.buffer);
}
escribirConsecutivos() {
    this.consecutivosProd.innerText = `Consecutivos: ${this.inRow}`;
}

async cambiarTextoEstado(texto) {
    await sleep(300); //dormimos la ejecución
    this.fieldState.innerText = `${texto}`;
}
get inRow() {
    return this._inRow;
}
set inRow(in_inRow) {
    this._inRow = in_inRow;
}

```

```
addConcurrency() {
    this._inRow++;
}
}

class Consumidor {
    constructor(buffer) {
        this.buffer = buffer;
        this.indexCons = 0; //Se reinicia al llegar a 25
        this._inRow = 0;
        this.espacioLiberado = "100";
        this.fieldState = document.getElementById("estadoConsumidor");
        this.consEstSleep = document.getElementById("consEstSleep"); //Representación de estado actual
        this.consEstAccesing = document.getElementById("consEstAccesing");
        //Representación de estado actual
        this.consEstCons = document.getElementById("consEstCons"); //Representación de estado actual
        this.consecutivosCons = document.getElementById("consecutivosCons");
        //Representación de estado actual
    }

    async consumir() {
        //this.consecutivosProd.innerText = `Consecutivos: ${this.inRow}`;
        removeAddClass(this.consEstSleep, "elementoEstado", "elementoEstadoActivo");

        await this.cambiarTextoEstado("Accediendo a buffer");
        removeAddClass(this.consEstSleep, "elementoEstadoActivo", "elementoEstado");
        /* Accesando */
        removeAddClass(
            this.consEstAccesing,
            "elementoEstado",
            "elementoEstadoActivo"
        );
        if (!this.buffer.includes("🍷")) {
            await this.cambiarTextoEstado("Durmiendo sin producto");
            removeAddClass(
                this.consEstAccesing,
                "elementoEstadoActivo",
                "elementoEstado"
            );
            //no se puede consumir
            removeAddClass(
                this.consEstSleep,
                "elementoEstado",
```

```

        "elementoEstadoActivo"
    );
    return;
}

await this.cambiarTextoEstado("Consumiendo");
removeAddClass(
    this.consEstAccesing,
    "elementoEstadoActivo",
    "elementoEstado"
);
//consumido
this.buffer[this.indexCons] = this.espacioLiberado;
document.getElementById(`spaceBuffer_${this.indexCons}`).innerText =
    this.espacioLiberado;
removeAddClass(this.consEstCons, "elementoEstado", "elementoEstadoActivo");
this.addConcurrency();
this.escribirConsecutivos();
await this.cambiarTextoEstado("Durmiendo");
removeAddClass(this.consEstCons, "elementoEstadoActivo", "elementoEstado");
//dormido
removeAddClass(this.consEstSleep, "elementoEstado", "elementoEstadoActivo");
this.indexCons = this.indexCons >= 24 ? 0 : this.indexCons + 1;
//console.log(this.buffer);
}
escribirConsecutivos() {
    this.consecutivosCons.innerText = `Consecutivos: ${this.inRow}`;
}

async cambiarTextoEstado(texto) {
    await sleep(300); //dormimos la ejecución
    this.fieldState.innerText = `${texto}`;
}
get inRow() {
    return this._inRow;
}
set inRow(in_inRow) {
    this._inRow = in_inRow;
}
addConcurrency() {
    this._inRow++;
}
}

function removeAddClass(domElement, remove, add) {

```

```
domElement.classList.remove(remove);
domElement.classList.add(add);
}

let prodCons;
let canceled = false;

async function initRuntime() {
  if (canceled) {
    return;
  }
  //await sleep(400); //dormimos la ejecución
  let actor = prodCons.decideAction();
  let steps = Math.random() * 1;
  //console.log(actor);
  if (actor instanceof Productor) {
    for (let index = 0; index < steps; index++) {
      await actor.producir();
    }
    window.requestAnimationFrame(initRuntime);
    return; //Finalización
  }
  for (let index = 0; index < steps; index++) {
    //es consumidor
    await actor.consumir();
  }
  //Repetimos el proceso
  window.requestAnimationFrame(initRuntime);
}

function reInitRuntime() {
  //Solo reiniciará si no esta cancelado
  if (!canceled) {
    alert("No se puede reiniciar si no se ha cancelado la ejecución");
    return;
  }
  canceled = false;
  initRuntime();
}

function sleep(ms) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}

document.addEventListener("DOMContentLoaded", (event) => {
```

```
prodCons = new ProductorConsumidor();

initRuntime();
window.addEventListener("keydown", (event) => {
  console.log(event.key);
  switch (event.key) {
    case "Escape": {
      canceled = true;
      break;
    }
  }
});
});
```

f) Realice un video con la demostración del programa funcionando, súbalo en algún sitio en la nube y no olvide proporcionar permisos públicos de visualización (si no se puede ver el video solo tendrá la mitad de los puntos de esta actividad).

g) Incluya el enlace al video en su reporte.

https://drive.google.com/file/d/193gk9rZWzJqHYhDy6ScsO_6CjJv-burd/view?usp=sharing

h) Genere una conclusión sobre lo que ha aprendido en esta actividad.

El problema del productor y consumidor es bastante útil para explicar la exclusión mutua de manera sencilla, desde luego que el problema desde esta perspectiva es bastante simple y no contempla muchas características de los sistemas actuales como un sistema de gestión de prioridades con referencia a la importancia de las tareas (no solo en su llegada), además, es una actividad que por si sola permite a quien la desarrolle trabajar en como organizar diferentes instancias para trabajar en conjunto siguiendo un conjunto de reglas que involucran el estado de otras instancias realizando la misma actividad